CS 471/571 Spring 2016

Lecture 2
Outline

• Network programming in C
• Network Terminology
• Network performance metrics
UDP Echo Server

#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <ctype.h>
int main(int argc, char *argv[]) {
    int sockfd;
    struct sockaddr_in server;
    struct sockaddr_in client;
    char msg[100];
    unsigned int clen;
    int len;
    int i;

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    memset(&server, 0, sizeof(struct sockaddr_in));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_port = htons(12000);
UDP Echo Server

bind(sockfd, (struct sockaddr *) &server, sizeof(struct sockaddr_in));

clen = sizeof(client);
len = recvfrom(sockfd, msg, 100, 0, (struct sockaddr *) &client, &clen);

for (i = 0; i < len; i++){
    msg[i] = toupper(msg[i]);
}
sendto(sockfd, msg, len, 0, (struct sockaddr *) &client, clen);

close(sockfd);
UDP Echo Client

//includes not shown
int main(int argc, char *argv[]) {
    int sockfd;
    struct sockaddr_in server;
    unsigned int slen;
    int len;
    char msg[100];

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    memset(&server, 0, sizeof(struct sockaddr_in));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_port = htons(12000);
UDP Echo Client

slen = sizeof(server);
len = strlen(argv[1]);
sendto(sockfd, argv[1], len, 0, (struct sockaddr *) &server, slen);
len = recvfrom(sockfd, msg, 100, 0, NULL, NULL);
msg[len] = '\0';
printf("%s\n", msg);
close(sockfd);
//includes not shown
int main(int argc, char *argv[]) {
    int sockfd1, sockfd2;
    struct sockaddr_in server;
    struct sockaddr_in client;
    char msg[100];
    unsigned int clen;
    int len;
    int i;

    sockfd1 = socket(AF_INET, SOCK_STREAM, 0);

    memset(&server, 0, sizeof(struct sockaddr_in));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_port = htons(12000);
TCP Echo Server

bind(sockfd1, (struct sockaddr *) &server, sizeof(struct sockaddr_in));

listen(sockfd1, 5);

sockfd2 = accept(sockfd1, (struct sockaddr *) &client, &clen);

len = recv(sockfd2, msg, 100, 0);
for (i = 0; i < len; i++)
    msg[i] = toupper(msg[i]);

send(sockfd2, msg, len, 0);

close(sockfd2);
close(sockfd1);
}
TCP Echo Client

```c
int main(int argc, char *argv[]) {
    int sockfd;
    struct sockaddr_in server;
    char msg[100];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    memset(&server, 0, sizeof(struct sockaddr_in));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_port = htons(12000);
```
TCP Echo Client

connect(sockfd, (struct sockaddr *) &server, sizeof(struct sockaddr_in));
send(sockfd, argv[1], strlen(argv[1]), 0);
int len = recv(sockfd, msg, 100, 0);
msg[len] = '\0';
printf("%s\n", msg);
close(sockfd);
INADDR_ANY (Server Side)

```c
memset(&server, 0, sizeof(struct sockaddr_in));
server.sin_family = AF_INET;
//server.sin_addr.s_addr = inet_addr("127.0.0.1");
server.sin_addr.s_addr = INADDR_ANY; //Server accepts messages on any interface
server.sin_port = htons(12000);
```
Client Connects to Remote Server

```c
server.sin_family = AF_INET;
//server.sin_addr.s_addr = inet_addr("127.0.0.1");
server.sin_addr.s_addr = inet_addr("138.49.30.14");
server.sin_port = htons(12000);
```
Client Connects to Server Given Symbolic Name

host = gethostbyname(argv[2]);

memset(&server, 0, sizeof(struct sockaddr_in));
memcpy(&server.sin_addr, host->h_addr_list[0], host->h_length);
server.sin_family = AF_INET;
//server.sin_addr.s_addr = inet_addr("127.0.0.1");
server.sin_port = htons(12000);
sockaddr

struct sockaddr {
    unsigned short   sa_family;
    char              sa_data[14];
};
sockaddr_in

struct sockaddr_in {
    short sin_family;
    unsigned short sin_port;
    struct in_addr sin_addr;
    char sin_zero[8];
};

struct in_addr {
    unsigned long s_addr;
};
struct hostent {
    char *h_name;    // official name of host */
    char **h_aliases;    // alias list */
    int h_addrtype;    // host address type */
    int h_length; // length of address */
    char **h_addr_list;    // list of addresses */
}
#define h_addr h_addr_list[0] /* for backward compatibility */
TCP Echo Client 2

#define MAXMESSAGE 1024

int main(int argc, char *argv[]) {
    int sockfd;
    struct sockaddr_in server;
    struct hostent *host;
    char msg[MAXMESSAGE];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    memset(&server, 0, sizeof(struct sockaddr_in));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = inet_addr(argv[1]);
    server.sin_port = htons(12000);
int ret = connect(sockfd, (struct sockaddr *) &server, sizeof(struct sockaddr_in));
if (ret == -1) { printf("errno %d\n", errno); exit(1);}

int i = 2;
while (i < argc) {
  sprintf(msg, "%s\n", argv[i]);
  send(sockfd, msg, strlen(msg), 0);
  int len = recv(sockfd, msg, 1024, 0);
  msg[len] = '\0';
  printf("%s", msg);
  sleep(1);
  i++;
}
sprintf(msg, "\n");
send(sockfd, msg, strlen(msg), 0);
close(sockfd);
int main(int argc, char *argv[]) {
    int sockfd1, sockfd2;
    struct sockaddr_in server;
    struct sockaddr_in client;
    unsigned int clen;

    sockfd1 = socket(AF_INET, SOCK_STREAM, 0);

    memset(&server, 0, sizeof(struct sockaddr_in));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(12000);
TCP Echo Server 2

```c
bind(sockfd1, (struct sockaddr *) &server, sizeof(struct sockaddr_in));

listen(sockfd1, 5);

while (1) {
    sockfd2 = accept(sockfd1, (struct sockaddr *) &client, &clen);
    serveClient(sockfd2);
}

close(sockfd1);
```
void serveClient(int sock) {
    int len;
    int i;
    char line[MAXMESSAGE];

    readline(line, sock);
    while (strcmp(line, "\n") != 0) {
        len = strlen(line);
        for (i = 0; i < len; i++)
            line[i] = toupper(line[i]);
        send(sock, line, len, 0);
        readline(line, sock);
    }
    close(sock);
}
void readline(char line[], int sock) {
    // no error checking
    int i = 0;
    char c;
    int len;
    len = recv(sock, &c, 1, 0);
    while (len == 1 && c != '\n') {
        line[i] = c;;
        i++;
        len = recv(sock, &c, 1, 0);
    }
    line[i] = '\n';
    line[i+1] = '\0';
pthread_t t;

while (1) {
    sockfd2 = accept(sockfd1, (struct sockaddr *) &client,
                        &clen);
    pthread_create(&t, NULL, serveClient, (void *) sockfd2);
}
void *serveClient(void *s) {
    int len;
    int i;
    char line[MAXMESSAGE];
    int sock = (int) s;

    readline(line, sock);
    while (strcmp(line, "\n") != 0) {
        len = strlen(line);
        for (i = 0; i < len; i++)
            line[i] = toupper(line[i]);
        send(sock, line, len, 0);
        readline(line, sock);
    }
    close(sock);
    return NULL;
}
Layered Architecture (Internet)

Application
  http, ftp, email, etc.

Transport
  program-to-program communication
  unit of communication: segment

Network
  end system-to-end system communication
  routing
  unit of communication: packet

Link
  neighbor-to-neighbor communication
  unit of communication: frame

Physical
  bit encoding on a physical media
  unit of communication: bit
Layered Architecture (OSI)

**Presentation Layer**
- Data representation

**Session Layer**
- Session maintenance including data recovery
Protocols

• Format and order of messages sent and received
• Actions taken when messages are sent and received
• In computer networks protocols define the way peer layers communicate
• In computer networks protocols implement services for higher layers based in the services provided by lower layers
Elements of the Internet

- PC
- server
- wireless laptop
- smartphone
- wireless links
- wired links
- router

- mobile network
- home network
- institutional network
- global ISP
- regional ISP
Elements of the Internet

- End Systems
- Access Networks
- Communication Links
- Network Core
Packet Switching

• Packet
• Store and forward
• Forwarding table
Performance Metrics

- Bandwidth (Data Rate)
- Transmission delay
- Propagation delay
Packet Delay

- Processing delay
- Queuing delay
- Transmission delay
  - Bits sent/Data rate
- Propagation delay
  - Depends on transmission medium and distance between the sender and the receiver
  - Distance/Propagation speed
  - Propagation speed
    - Between $2 \times 10^8$ meters/second and $3 \times 10^8$ meters/second
Packet Delay

• Total packet delay in a node
  – Processing delay (procD) + queuing delay (queD) +
    Transmission delay (tranD) + propagation delay (propD)

• End-to-end delay
  – Suppose the packet traverses N nodes for the
    source to the destination
  – One way end-to-end delay
    • \( \sum_{i=1}^{N} (\text{procD}_i + \text{queD}_i + \text{tranD}_i + \text{propD}_i) \)
Queuing Delay and Packet Loss

• Arrival rate (bits/second)
  – Modeled with statistical measures
  – The book uses the formula $L_a$ where $L$ is the size of the packets (in bits), $a$ is the arrival rate of packets

• Traffic intensity
  – Arrival rate/Data rate
  – When traffic intensity $> 1$ queuing delay occurs
  – Traffic intensity varies over time

• Packet loss
Average Queuing Delay
Throughput

• Total bits sent/time measured
  – Instantaneous
  – Average

• Bottleneck links
  – The throughput cannot be greater than lowest throughput link
Throughput

The server sends bits (fluid) into a pipe that can carry fluid at rate $R_s$ bits/sec. The pipe is limited by the link capacity $R_c$ bits/sec.
Throughput

Maximum Throughput

Throughput

Offered Load
Other Performance Metrics

• Round trip delay
  – Packet sent
  – Acknowledgement (ACK) received

• (Round trip delay) * (data rate)
  – This formula is useful because it indicates the number of bits transmitted before an ACK can be received
Practice Problems

• Suppose you want to move a 200MB file from machine A to machine B. Machine A is connected to machine B by a fiber link that is 10 km long. The data rate is 100 Mbps and the propagation speed is $2 \times 10^8$ m/s.

• (a) If you can transmit continuously, how long (measured from the time the first bit is sent) will it take for the last bit to arrive at B?
Practice Problems

• Now suppose the file is broken into packets of 4096 bytes (ignore packet overhead).

• (b) If the protocol used requires each packet to be acknowledged (ACK) before the next packet is sent and the processing time to generate an ACK and to transmit the ACK is negligible, how long will it take the send the file measured from the time the first bit is sent to the time the last ACK arrives.

• (c) Repeat the above problem except assume an ACK must only be sent once for every 20 packets.